

Replica Management in the Data Grid

Ann Chervenak and Chuck Salisbury

February 25, 2000

1 Introduction

The *Computational Grid* is an emerging infrastructure that combines massive storage systems, high-performance and parallel networks, high-performance microprocessors, parallel computers, communication protocols, distributed system software, and security mechanisms. The Grid allows applications to share computing resources and data sets that are distributed over the wide area.

The *Data Grid* focuses on the management of storage resources and large datasets in a Grid environment. In an increasing number of scientific disciplines such as global climate change studies, high energy physics, and computational genomics, large data collections are emerging as important community resources. These volume of these data collections is already measured in terabytes and will soon total petabytes. Components of the Globus Data Grid infrastructure include:

- Standard interfaces to a wide variety of storage systems and devices
- A service for maintaining metadata, or attributes that describe particular files, collections of files, and storage systems. Users can query datasets based on their attributes.
- A service for creating and managing replicas of datasets

In this paper, we describe the replica management architecture of the Data Grid. This service is responsible for managing complete and partial copies of data sets. Replica management is an increasingly important issue for a number of scientific applications. For example, consider a data set that contains one petabyte of experimental results for a particle physics application. While the complete data set may exist in one or possibly several physical locations, it is likely that many universities, research laboratories or individual researchers will have insufficient storage to hold a complete copy. Instead, they will store copies of the most relevant portions of the data set on local storage for faster access.

Services provided by a replica management system include:

- creating new copies of a complete or partial data set
- registering these new copies in a *Replica Catalog*
- allowing users and applications to query the catalog to find all existing copies of a particular file or data set
- selecting the “best” replica for access based on storage and network performance predictions provided by a Grid information service

A fundamental component of the replica management architecture is the *Replica Catalog*, which provides *mappings* between logical names for files and collections and the storage system locations of one or more replicas of these objects.

Note that the Replica Catalog should not contain any *semantic* information that describes the contents of data files and collections. Semantic information includes objects and attributes related to the contents and structure of data sets. Such information should be stored in a separate *Metadata Catalog*. Several scientific disciplines such as climate modeling, particle physics, and neurobiology have well-defined Metadata Catalogs. (Add references.) Users or applications can query these Metadata Catalogs to identify files and collections with desired attributes. The Metadata Catalog produces a list of names for the requested files and collections; depending on the catalog, these names may be direct pointers into the Replica Catalog, or a conversion step may be necessary before consulting the Replica Catalog to determine the physical locations of the chosen files.

In the remainder of this paper, we present details of the design of the Replica Catalog architecture. We illustrate the architecture with an example implementation using LDAP, the Lightweight Distributed Access Protocol. Finally, we discuss potential extensions to the design of the replica management service.

2 The Replica Catalog Architecture

As mentioned above, the purpose of the Replica Catalog is to provide mappings between logical names for files or collections and one or more copies of the objects on physical storage systems.

The Replica Catalog architecture consists of a minimal set of objects that characterize logical files and logical collections as well as provide mapping information between logical names and physical storage locations. These objects include: *replica catalog*, *logical collection*, *logical file*, and *replica*. This set of objects may eventually be expanded.

In the discussion below, we describe the hierarchy of objects in the Replica Catalog. We present an example catalog implementation for portions of a climate modeling data set, shown in Figure 1. This example Replica Catalog is implemented as an LDAP directory.

2.1 The Replica Catalog Object

At the top level of the object hierarchy shown in Figure 1 is the `replicaCatalog` object. A Replica Catalog contains mappings between names of logical collections and logical files and physical locations of one or more copies of files.

Although Figure 1 shows a single `replicaCatalog` object, a data grid may (and indeed typically will) contain multiple replica catalogs. For example, researchers in different scientific disciplines might maintain separate replica catalogs to manage relevant data sets. It is possible to create hierarchies of replica catalogs to impose a directory-like structure on related logical collections. In addition, the replica manager can perform access control on an entire replica catalog as well as on individual logical files.

Next, we present an object definition for a particular implementation of the `replicaCatalog` object in an LDAP directory within the Globus grid computing environment. This simple object contains an attribute that allows the creator of a catalog to describe it:

```
GlobusReplicaCatalog ObjectClass
    SUBCLASS OF GlobusTop
    SUBCLASS OF GlobusGroup
    RDN = rc (replicaCatalog)
```

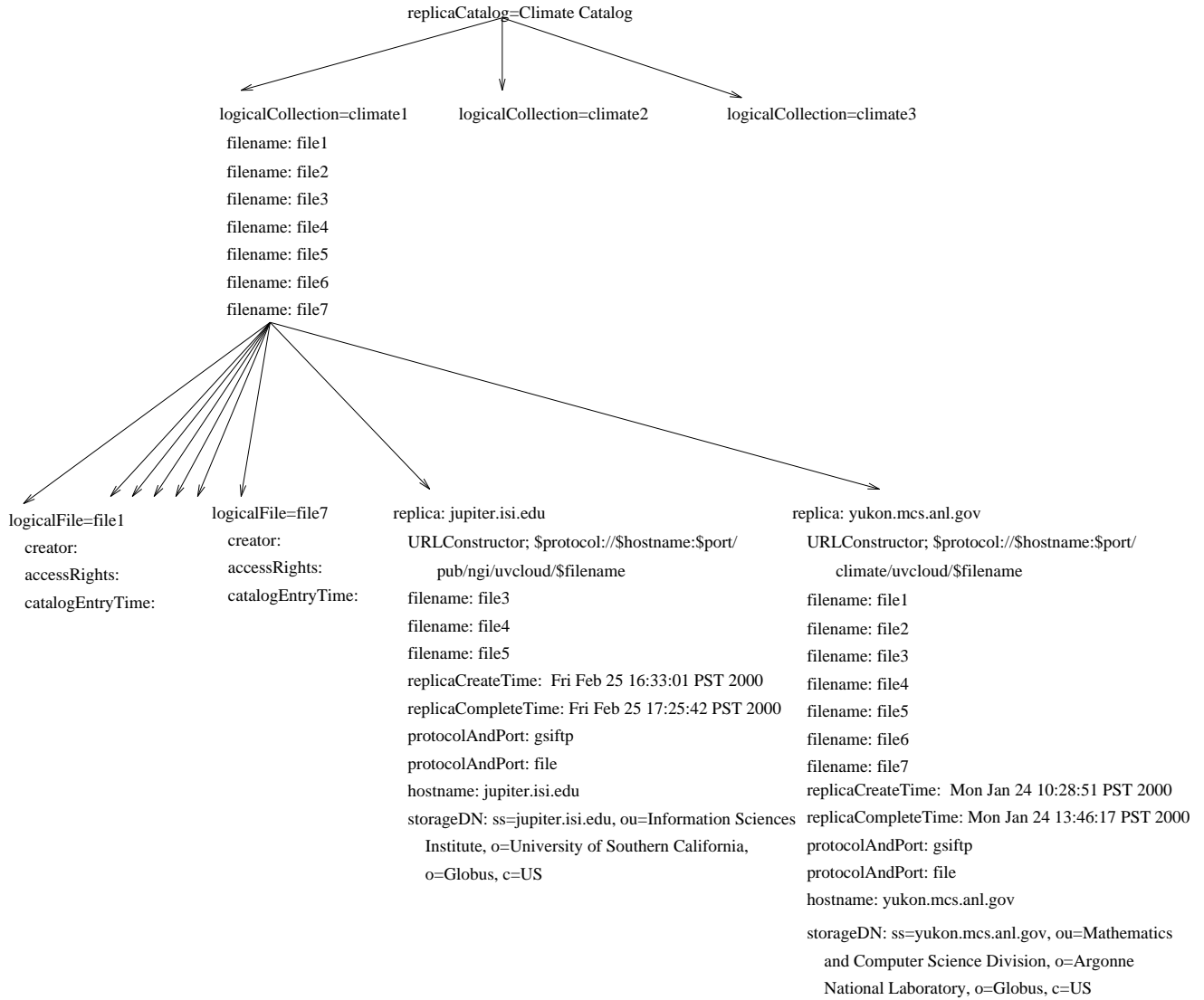


Figure 1: Shows the Replica Catalog architecture.

```

CHILD OF {
}
MAY CONTAIN {
    description :: cis,          # description of catalog
}

```

2.1.1 The Logical Collection Object

At the top level of the replica catalog are one or more logicalCollection objects. A logical collection object groups together logical files for the purpose of managing replication more efficiently. Typically, a research community or an application would group together logically-related files and register them as a collection in the replica catalog. The catalog is not concerned with the contents of files or collections. It is only concerned with how the files in the collection are replicated on different storage systems, as explained in the *replica* object description below.

We require that all logical files registered in the Replica Catalog belong to a single logical collection object.

The implementation of a logicalCollection object in an LDAP directory in the Globus environment is shown below. The logicalCollection object must contain a complete list of the names of all logical files in the collection. Thus, one operation on the replica catalog is to **return the names of all logical files associated with a logical collection**.

The logicalCollection object also contains optional attributes for describing the logical collection, including information about the creator, creation time and access rights for the catalog.

```

GlobusReplicaLogicalCollection ObjectClass
SUBCLASS OF GlobusTop
RDN = lc (logicalCollection)
CHILD OF {
    GlobusReplicaCatalog
}
MUST CONTAIN {
    filename :: cis,          # name of logical file in collection
}
MAY CONTAIN {
    description :: cis,      # description of logical collection
    creator :: cis,         # of logical collection
    accessRights :: cis,     # for accessing logical collection
    catalogEntryTime :: cis, # of logical file collection
}

```

2.1.2 The Logical File Object

Logical files are entities with globally unique names that may have one or more physical instances. For each logical file in a collection, there is exactly one logicalFile object in the replica catalog. Given this structure, an LDAP implementation of the replica catalog supports queries that **return the full distinguished name (DN) of all logical files in a specified logical collection**.

The logicalFile object in a Globus LDAP implementation of a Replica Catalog is shown below. The object contains optional attributes describing the logical file, including the creator of the entry in the replica catalog, time of entry and access rights.

```

GlobusReplicaLogicalFile ObjectClass
    SUBCLASS OF GlobusTop
    RDN = rf (replicaLogicalFile)
    CHILD OF {
        GlobusReplicaCatalog
    }
    MAY CONTAIN {
        description :: cis,           # description of logical file
        creator :: cis,              # of logical file
        accessRights :: cis,         # for accessing logical file
        catalogEntryTime :: cis,     # of logical file object
    }

```

2.2 The Replica Object

Beneath the logical collection object are one or more *replica* objects. The replica object contains the information required to map logical names for files and collections to physical storage locations or URLs.

The replica object provides information about a complete or partial copy of a logical collection. The replica object explicitly lists all files from the logicalCollection object that are contained in the replica; so, the replica object can specify either partial or complete collections. Note that all logical files in a replica are stored on the same physical storage system. A logicalCollection object may have an arbitrary number of replicas, each of which contains a (possibly overlapping) subset of the files in the collection. Using replica objects, we can easily implement logical collections that span multiple physical storage systems.

In the example shown in Figure 1, the logicalCollection called climate1 has two replicas: a partial collection that contains file3, file4 and file5 on the storage system jupiter.isi.edu, and a complete collection on the storage system yukon.mcs.anl.gov.

The replica catalog supports a number of operations using the replica object:

- **use hostname, protocol, port and filename attributes to construct a URL for the desired logical file using the grammar specified in the URLconstructor attribute**
- **find all replicas of a specified logical file by searching the filename attributes of replica objects**
- **using Grid information systems, estimate data transfer performance for each replica of a desired file or collection; select the replica with the best predicted performance**

An example replica object in our LDAP implementation has the following attributes:

```

GlobusReplicaInfo ObjectClass
    SUBCLASS OF GlobusTop
    RDN = re (replica)
    CHILD OF {

```

```

        GlobusReplicaLogicalFile
    }
    MUST CONTAIN {
        URLConstructor :: cis,                # used to construct
                                              # URLs of files
        completeTime :: cis,                  # complete time/pending
        filename :: cis,                      # name of logical file
    }
    MAY CONTAIN {
        storageDN :: dn,                      # DN of storage system
                                              # where files are stored
        createTime :: cis,                   # start time of replica creation
        creator :: cis,                      # creator of replica
        originalURL :: cis,                  # URL of originating
                                              # file instance
        logstreamDN :: dn,                   # DN of entry in log file
        protocolAndPort :: cis,              # access protocol and
                                              # port for storage system
        hostname :: cis,                     # hostname of storage system
    }

```

Each replica of a collection must explicitly name all the logical files in that partial replica, using the filename attribute.

The URLConstructor attribute specifies a grammar that the replica catalog uses to generate a complete URL, which is required by Globus to read or write individual files. Typically, the URLconstructor specifies a format for generating a URL by combining the protocol specification, hostname and port number of the storage device (available from the storageSystem object), the string specified by the URLconstructor, and the name of the file on the target storage system.

The replica object contains an optional attribute called storageDN which contains a pointer to a StorageSystem object; when used in a Globus environment, the StorageSystem object resides in the Grid information service, called MDS, and characterizes the system where the file is stored, including access protocols, system performance and configuration.

The replica object also contains optional attributes. The object can explicitly specify the access protocol for accessing the storage system and the storage system's hostname and port number. (If port number is not specified, the default port number for the chosen protocol is assumed.) If the Replica Catalog is part of a Globus grid environment, then these attributes can be found in the MDS information service. However, we include them as optional attributes of the replica object to allow the Replica Catalog to be used independently of Globus and MDS.

Additional attributes of the replica object include times indicating when a replica creation was initiated and completed; the name of the replica's creator; the URL from which the replica was copied; and a pointer into a log file that records replica operations. Of these attributes, only completeTime is required.

3 Potential Extensions

We have described a set of objects to implement replica management. The basic capabilities outlined so far could be extended in several ways. In this section, we discuss some of the potential extensions to

the catalog architecture. We welcome comments from the user community regarding the desirability of these extensions.

3.1 Views

The proposed design provides an information location service using a catalog structure that is the same for all users. Access to entries in the catalog can be controlled using LDAP facilities. However, one possible requirement is to increase the amount of local control, providing local or even personal *views* of the distribution of data.

A personal view could allow a user to record the locations of a set of files being used repeatedly. This would avoid the need to query the metadata catalog for logical collection and file names, search the replica catalog for the corresponding locations, and then select from among the locations. The personal view could associate a logical name with a set of file instance URLs. This would also provide users with the ability to easily locate data that has been placed in different logical collections. Each user's set of personal views could be independent of the views of other users.

A structure that provides local views might be used in an environment providing increased local control over data access. In addition to the global catalog, site catalogs could be used to record the location of data available to users at a particular site, but not generally accessible outside the site. Maintenance and control of the catalog would be site responsibilities. The local replica catalog would contain links to replica catalogs at sites which share data, as well as to any replica catalogs which are globally known. Providing this local control would probably come at the cost of increased complexity for locating data not recorded in the site catalog.

3.2 Mapping file names to sub-directories

The proposed design implicitly assumes that all files in a collection will be stored in a single directory. Specifically, the URL constructor contained in a replica object provides a single path and a simple language for mapping a logical file name into the name of the file instance. It might be the case that a system administrator would place a single collection with a large number of files into different subdirectories to ease storage management. When a logical collection is physically stored on multiple directories, there must be some mechanism for determining the correct sub-directory for each logical file.

One approach would be to extend the mapping language to associate a logical file name with path information as well as a file instance name. The requirements for such a mapping language have not been defined, but would probably be based on the logic used by system administrators to distribute the files across several subdirectories.

3.3 Attribute Extensibility

It can be difficult to determine which descriptive information belongs in a metadata catalog, and which in the replica catalog. Our design places in the replica catalog only that information associated with mapping logical file and collection names to physical storage locations. However, additional information may be desirable.

Stated in more general terms, the data grid architecture does not define the semantics of replicas. In our catalog architecture, a replica is a user-asserted correspondence between two physical files. Replicas specified in the catalog *may* be byte-for-byte copies of one another, but this is not required by the Replica Manager. It may be desirable to allow users to place in the replica catalog attributes that define the semantics of replicas.

As an example, consider data that is stored in big endian and little endian formats at different locations. In a grid environment, a user may have access to both types of machines and want to determine all locations of the data prior to selecting both a data source and a compute location. Another example is data that may be stored in one file by temporal coordinates and in another by spatial coordinates. While the files contain the same information, an application's performance may differ for the two files. In both of these examples, it may be desirable to allow the users to place additional attributes in the replica catalog to assert that files are replicas.

4 Summary

Replica management is an increasingly important function for managing individual files and collections of files for scientific communities.

We have presented an overview of the architecture for replica management in the Data Grid. One important component of this architecture is the Replica Catalog, which provides mappings between logical names of files and collections and the physical locations where these objects are stored. The replica catalog architecture consists of four objects: *replica catalogs*, *logical collections*, *logical files* and *replicas*. Using these simple objects, we are able to provide many essential replica management services. The catalog allows users and applications to register logical files and collections in the replica catalog, as well as registering any number of replicas of subsets of a specified logical collection. In addition, we can perform a number of important queries on the replica catalog, including:

- find all logical file names associated with a logical collection
- find all distinguished names of logical files in a collection
- find all replicas of a logical file or collection
- construct a URL for a replica of a logical file or collection

Finally, in association with a Grid computing information service, we can estimate the performance of various replicas and choose the best replica for a given data transfer.

We have also discussed a set of possible extensions to our replica catalog, including allowing arbitrary views of replicas; mapping filenames to more complex hierarchies of directories; and extending attributes associated with replicas to allow users more control over the definition and selection of replicas.

We welcome comments on our proposed Replica Catalog architecture and possible extensions. Please send email to annc@isi.edu and salisbur@mcs.anl.gov.